COMP3258

Functional Programming

Tutorial 9: Equational Reasoning and Structural Induction

Warm-up (1)

Given the definition of not show that

```
not (not x) = x
```

```
not True  = False
not False = True
```

Warm-up (2)

Given the definition of SAE* and eval, show that

```
eval (Add e1 (Add e2 e3)) = eval (Add (Add e1 e2) e3)
```

```
data SAE = Number Integer
         | Add SAE SAE
```

```
eval :: SAE -> Integer
eval (Number n)
eval (Add e1 e2) = eval e1 + eval e2
```

*https://vesely.io/teaching/CS4400f20/l/06/06.pdf

```
eval (Add e1 (Add e2 e3))
= {applying eval}
eval e1 + eval (Add e2 e3)
= {applying eval}
eval e1 + (eval e2 + eval e3)
= {assoc of (+)}
(eval e1 + eval e2) + eval e3
= {unapplying eval}
eval (Add e1 e2) + eval e3
= {unapplying eval}
eval (Add (Add e1 e2) e3)
```

Question 1

Given the definition of map and (.), show that

```
map f (map g xs) = map (f . g) xs
```

```
map f [] = []
map f (x:xs) = f x : map f xs

(f . g) x = f (g x)
```

induction on xs

Base case:

map f (map g [])
= {applying inner map}
map f []
= {applying map}
[]

Inductive case:

map f (map g (x:xs))
= {applying inner map}
map f (g x : map g xs)
= {applying outer map}
f (g x) : map f (map g xs)
= {induction hypothesis}
f (g x) : map (f . g) xs
= {unapplying (.)}
(f . g) x : map (f . g) xs
= {unapplying map}
= map (f . g) (x : xs)

Question 2

Given the definition of all and replicate, show that

```
all (== x) (replicate n x) = True
```

```
all p []     = True
all p (x:xs) = p x && all p xs

replicate 0 x = []
replicate n x = x : replicate (n - 1) x
```

induction on n

Base case:

```
all (== x) (replicate 0 x)
= {applying replicate}
all (== x) []
= {applying all}
True
```

Inductive case: n = m + 1

```
all (== x) (replicate (m + 1) x)
= {applying replicate}
all (== x) (x : replicate m x)
= {applying all}
(== x) x && all (== x) (replicate m x)
= {applying ==}
True && all (== x) (replicate m x)
= {induction hypo}
True && True
= {applying &&}
True
```

Question 3

Given the definition of (++), show that

```
1) xs ++ [] = xs
2) xs ++ (ys ++ zs) = (xs ++ ys) ++ zs
```

```
[] ++ ys          = ys
(x : xs) ++ ys = x : (xs ++ ys)
```

1) xs ++ [] = xs

induction on xs

Base case:

```
[] ++ []
= {applying (++)}
[]
```

Inductive case:

```
(x : xs) ++ []
= {applying (++)}
x : (xs ++ [])
= {induction hypothesis}
x : xs
```

2) xs ++ (ys ++ zs) = (xs ++ ys) ++ zs

induction on xs

Base case:

```
[] ++ (ys ++ zs)
= {applying (++)}
ys ++ zs
= {unapplying (++)}
([] ++ ys) ++ zs
```

Inductive case:

```
(x:xs) ++ (ys ++ zs)
= {applying (++)}
x : (xs ++ (ys ++ zs))
= {induction hypothesis}
x : ((xs ++ ys) ++ zs)
= {unapplying (++)}
(x : (xs ++ ys)) ++ zs
= {unapplying (++)}
((x : xs) ++ ys) ++ zs
```

Question 3

Given the definition of (++), take and drop, show that

```
take n xs ++ drop n xs = xs
```

```
take 0 xs = []
take (n + 1) [] = []
take (n + 1) (x : xs) = x : take n xs

drop 0 xs = xs
drop (n + 1) [] = []
drop (n + 1) (_ : xs) = drop n xs
```

Induction on n xs

Base case: n = 0, xs = []

```
take 0 [] ++ drop 0 []
= {applying take, drop}
[] ++ []
= {applying ++}
[]
```

Base case: n = m + 1, xs = []

```
take (m+1) [] ++ drop (m+1) []
= {applying take, drop}
[] ++ []
= {applying ++}
[]
```

Base case: n = 0, xs = x:xs

```
take 0 (x:xs) ++ drop 0 (x:xs)
= {applying take, drop}
[] ++ []
= {applying ++}
[]
```

Inductive case: n = m + 1, xs = x:xs

```
take (m+1) (x:xs) ++ drop (m+1) (x:xs)
= {applying take, drop}
(x : take m xs) ++ drop m xs
= {unapplying (++)}
x : (take m xs ++ drop m xs)
= {induction hypo}
x : xs
```

Question 4

Given the definition of Tree, show that
the number of leaves in a tree is always one greater than the number of nodes,
by induction on trees.

```
data Tree = Leaf Int | Node Tree Tree
```

First we define two functions

```
nodes  :: Tree -> Int
leaves :: Tree -> Int

nodes (Leaf _) = 0
nodes (Node l r) = 1 + nodes l + nodes r

leaves (Leaf _) = 1
leaves (Node l r) = leaves l + leaves r
```

Prove: (nodes t) + 1 = leaves t

Induction on t

Base case: t = Leaf n

(nodes (Leaf n)) + 1
= {applying nodes}
0 + 1
= {arithmetic calculation}
1
= {unapplying leaves}
leaves (Leaf n)

Inductive case: t = Node l r

nodes (Node l r) + 1
= {applying nodes}
1 + nodes l + nodes r + 1
= {permutation of addition}
((nodes l) + 1) + ((nodes r) + 1)
= {induction hypo}
leaves l + leaves r
= {unapplying leaves}
leaves (Node l r)